Ab Initio Quantum Chemistry on the Cray T3E Massively Parallel Supercomputer: II

C. P. SOSA,[1] J. OCHTERSKI,[1] J. CARPENTER,[1] M. J. FRISCH[2]

[1]*Cray Research / Silicon Graphics, Inc., 655 E. Lone Oak Drive, Eagan, Minnesota 55121*
[2]*Lorentzian, Inc., North Haven, Connecticut*

Received 17 September 1997; accepted 16 February 1998

ABSTRACT: Gaussian-94 is the series of electronic structure programs. It is an integrated system to model a broad range of molecular systems under a variety of conditions, performing its calculations from the basic laws of quantum chemistry. This new version includes methods and algorithms for scalable massively parallel systems such as the Cray T3E supercomputer. In this study, we discuss the performance of Gaussian using large number of processors. In particular, we analyze the scalability of methods such as Hartree–Fock and density functional theory (DFT), including first and second derivatives. In addition, we explore scalability for CIS, MP2, and MCSCF calculations. Scalability and speedups were investigated for most of the examples with up to 64 process elements. A single-point energy calculation (B3-LYP/6-311 + + G(3df,3p) was tested with up to 512 process elements. © 1998 John Wiley & Sons, Inc. J Comput Chem 19: 1053–1063, 1998

Keywords: Gaussian; massively parallel; performance; Cray T3E

Introduction

Parallel processing long has been recognized as a potentially powerful tool for faster computing. As the speed of single processors approach physical limitations, such as the speed of light, it becomes more difficult to improve performance based on single processors.[1] Today's traditional vector supercomputers have a clock cycle of 2.2 nanoseconds (ns) (Cray T90). The time that it takes light to travel 1 foot is about 1 ns. Given our current technology, electrical signals travel through 1 foot of flex cables (Cray T3E) or PC boards (T90) anywhere between 1.4 and 1.9 ns.[2] These physical limitations make using an ensemble of processors (either scalar, vector, or a combination of both) an attractive alternative to faster clock speeds for placing more computer power into one machine. Traditionally, computers that take a single se-

Correspondence to: C. P. Sosa

CCC 0192-8651 / 98 / 091053-11

quence of instructions and operate on a single sequence of data are usually referred as single instruction stream, single data stream (SISD).[3] Instructions (operations) are processed sequentially and data streams transfer from memory to functional units.

On the other hand, multiple-processor computers come in a variety of architectures.[4] Based on the control mechanism. Computers that operate under a centralized single control unit are referred as single instruction stream, multiple data stream (SIMD), in which the control unit sends the *same* instruction (operation) to all the processing units. Computer architectures in which each processor can issue instructions independently from one another are called multiple instruction stream, multiple data stream (MIMD). Address-space (memory) organization and interconnection network can also be used to further classify multiprocessor computer systems.

The Cray T3E, a second generation massively parallel supercomputer from Cray Research, was used to complete this study. It is a MIMD computer system with physically distributed, but globally addressable, memory. This new generation of computer systems has been recognized as a powerful tool in high-performance computing, but it has not been widely used in a production environment. One of the initial efforts parallelizing quantum chemistry codes was undertaken by Clementi and coworkers[5] where they experimented with the so-called loosely coupled array of processors (LCAP) system.[6] Recently, Harrison and Shepard [7] have reviewed several of the different techniques and attempts that chemists have carried out in the parallelization of *ab initio* computer codes.

In two previous studies, Turner et al.[8] reported the implementation of Gaussian on a cluster of workstations (from now on referred as part I). In part I of this work, a small cluster of workstations (a maximum of six Unix workstations) was used to study the parallel performance of single-point energies, gradients, and frequencies at the Hartree–Fock level only, and Ochterski et al.[9] presented its parallelization on Cray vector supercomputers. In a similar effort, Cooper and coworkers[10] implemented a parallel Hartree–Fock method for the calculation of two-electron integrals and a parallel version of density functional theory using the parallel virtue machine (PVM)[11] programming model. Their work was interfaced to a modified version of Gaussian-92. More recently, Scalmani and coworkers[12] have parallelized portions of Gaussian-94 using PVM and message passing interface (MPI).[13]

The Cray T3E includes features that are designed to deliver high-performance on scientific codes such as Gaussian. In this study, we extend the work presented in part I by reporting the parallel performance of methods such as density functional theory (DFT), Møller–Plesset theory up to second order (MP2), CI singles (CIS), and multiconfiguration self-consistent theory (MCSCF). In addition, we present, for the first time, Gaussian parallel performance on a massively parallel supercomputer. We hope that this will help spur the use of commercially available applications such as Gaussian on this next generation of supercomputers. The next section provides a brief overview of the Cray T3E architecture. The third section gives an overview on how Gaussian has been parallelized. The fourth section describes the memory distributed version of Gaussian. The fifth section discusses the performance of the examples studied in this report. The final section gives a brief summary.

Design Features in the Cray T3E Massively Parallel System

A major difference between traditional vector supercomputers and MPP machines is in the memory architecture.[14] Traditional vector supercomputers that use parallel vector processors (PVP) have one uniform shared-addressable memory among all the processors. For example, the Cray T90 has 32 processors, each with very rapid access to central memory. Any processor can read any word in memory with the same time delay. On the other hand, MPP systems, such as the Cray T3E used in this work, have distributed memory, so different words have different access times, depending on the relative locations of processors and memory. This type of memory architecture is referred as nonuniform memory access (NUMA). Each processor on the T3E is connected by a bidirectional three-dimensional (3D) torus system interconnect network.[15]

Three different types of Cray T3Es were used in this study; the Cray T3E-600, Cray T3E-900, and Cray T3E-1200, with a maximum of 2048 process elements (PEs). Each PE on the T3E-600, T3E-900, and T3E-1200 systems are composed of DEC Alpha EV5 (300 MHz), EV5.6 (450 MHz), and EV6 (600 MHz) RISC microprocessors, respectively.

Memory sizes used were from 128 MBytes to 512 Mbytes per PE. Peak performance levels of the Cray T3E (300 MHz), Cray T3E (450 MHz), and Cray T3E (600 MHz) are 600, 900, and 1200 MFlops per PE, respectively. For a more detailed study on a variety of performance measurements and hardware description see ref. 16.

Gaussian Parallelization Overview

Gaussian,[17] a connected series of programs, can be used for performing a variety of semiempirical, *ab initio*, and density functional theory calculations. Each program communicates with each other through disk files. All the individual programs are referred as links, where links are grouped into overlays.[18] In general, overlay 0 is responsible for starting the program, including reading of the input files. Once the route card is read, the proper set of overlays/options/links is selected for a particular run. Overlay 99 (`L9999`) terminates the run, and in most cases `L9999` produces a summary of the calculation (archive entry).

The Gaussian architecture on shared–addressable or distributed memory machines is basically the same; that is, each link is responsible for continuing the sequence of links by invoking the *exec*( ) system call to run the next link. Optimization of this process involves either decreasing the user process time or the elapsed time. The first step in optimizing a program is evaluating its overall performance and looking for computational bottlenecks. It is well known that, in molecular electronic calculations, the rate-limiting step is the computation of the two-electron integrals[19] and their derivatives. This has already been summarized in a previous study.[9] A brief description of that work will be presented here. Also, recently, Harrison and Shepard[7] have extensively reviewed parallel algorithms for self-consistent field (SCF) calculations.

In a self-consistent field (SCF) scheme, the two-electron integrals are part of the Fock matrix[20]:

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda}\sum_{\sigma} P_{\lambda\sigma}\left[(\mu\nu|\lambda\sigma) - \frac{1}{2}\times(\mu\lambda|\nu\sigma)\right] \quad (1)$$

where $H_{\mu\nu}$ represents the core Hamiltonian. μ, ν, λ, and σ are atomic orbital indices (AO). The quantities $(\mu\nu\,|\,\lambda\sigma)$ are two-electron repulsion integrals. In Gaussian, these quantities are computed once and stored or recomputed many times as needed, depending on the memory available and the algorithm chosen.

In density functional theory (DFT), eq. (1) can be rewritten by replacing the last term by the well-known exchange-correlation term F^{XC}[21]:

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda}\sum_{\sigma} P_{\lambda\sigma}(\mu\nu|\lambda\sigma) + F_{\mu\nu}^{XC} \quad (2)$$

Parallelization of the Fock matrix involves distributing batches of integrals among all the available processors. This approach is similar to the work of Cooper et al.[10] where they have parallelized the two-electron integral generation via the PRISM algorithm. They distribute batches of shell quartets that can easily be added to the Fock matrix. The main difference between their work and this is that we have chosen Linda as the parallel programming model. Cooper et al. used PVM.[10] The parallel structure of Fock matrix elements formation for Hartree–Fock and DFT are shown in Figure 1. At the end of the first loop over N_{PEs}, all the contributions to the Fock matrix have been computed, the last DO loop adds all the contributions together in a serial block of code.

Inside the loop over the number of PEs, several loops generate the drivers needed to compute integrals and the shell quartets to look at. Each task looks at $1/N_{PEs}$ of the shell quartets, discarding those that do not need to be done due to symmetry, cutoffs, or the fact that they have already been done. Then, the pertinent integrations are carried out. The last loop sums up the contributions to the Fock matrix, derivative matrices, or density matrices, depending on the type of integrals computed. In effect, the same approach is used to compute first and second two-electron integral derivatives and electrostatic potential integrals. Table I summarizes all the links that require evaluation of two-electron integrals and hence are capable of using multiple processors.

```
loop over NPEs
 loop over total angular momentum
  loop over degrees of bra and ket contraction
   do integrals for 1/NPEs of shell quartets
 end loop
 end loop
 add integral contributions to partial Fock matrix
end loop
loop over NPEs (serial code)
  add 1/NPEs Fock matrix contributions
end loop
```

FIGURE 1. Parallelization of PRISM.

TABLE I.
Linda Links in Gaussian-94 (Revision E.2)

Linda links	Description
L502	Closed- and open-shell SCF solution
L506	GVB solution
L508	Quadratically convergent SCF solution
L510	Multiconfiguration SCF solution
L602	One-electron properties
L703	Two-electron integral first- or second-derivative evaluation
L906	Direct and semidirect MP2 energies and gradients
L914	Calculates excited states using CI with single excitations
L1002	Coupled perturbed Hartree–Fock solution and contribution of coefficient derivatives to Hartree–Fock second derivatives
L1014	Coupled perturbed CI singles
L1110	Two-electron contribution to Fock matrix derivatives with respect to nuclear coordinates
L1112	Forms most of the terms in MP2 second derivatives

Distributed Memory Gaussian (Revision E.2)

The aforementioned parallelization scheme has been implemented in the latest public release of Gaussian. The implementation of two-electron repulsion integrals uses the PRISM algorithm.[22] PRISM has been parallelized using Linda. Linda is a model for parallel processing based on distributed data structures.[23] Processors and data objects communicate by means of the tuple space (TS) or Linda memory. Linda's virtual memory (shared–addressable memory-like) on the Cray T3E provides users with a simple and flexible programming environment.[24] Also, it is a model familiar to programmers on Cray vector systems. Linda creates a virtual shared memory that is shared logically by all the processors (workers in Linda notation or slave processors) on the T3E. The interaction between the tuple space (Linda's virtual memory) and the processors is carried out by four basic operations: *out*, *in*, *eval*, and *rd*. These operations add or remove data objects to or from the tuple space. *eval*, on the other hand, also forks processes to initiate parallel execution.

In the Linda implementation of the PRISM algorithm, the vast majority of the data needed by slave processors is passed through tuple space. Storing or retrieving different types of data from or to the tuple space is carried out by three routines, prlin1, prlin2, and prlin3. These routines are called by the driver prsmsu (caldsu for DFT calculations). In addition, prsmsu makes sure that the amount of memory available is consistent with the number of processors requested. The *eval* operation allows only a small number of simple data items to be passed in the argument list of the created process, thus, most of the data has to be exchanged via the tuple space. Once all the data are in the tuple space, PRISM is called. PRISM receives shell-pair data to form batches of brackets to compute two-electron integrals.[22]

In the just discussed algorithm (Fig. 1), each worker is assigned a batch of shell quartets to compute partial sums for every element of the Fock matrix. Each worker stores its results in tuple space. All processors are synchronized and, in the final step, the master processor sums the matrices stored in tuple space into its own Fock matrix. This strategy can be implemented for single-point energy calculations, derivatives of the energy,[8] CI singles, or MCSCF. This is a replicated data-type algorithm (the Fock matrix gets replicated on each processor) that has been implemented in a number of quantum chemistry codes.[7]

In the distributed memory version, MP2 calculations (single point) can also be performed using multiple processors. Parallelization of MP2 calculations has been discussed in previous studies. Watts and Dupuis[25] implemented in-core and out-of-core algorithms on shared–addressable memory computers. More recently, Limaye and Gadre[26] and Nielsen and Siedl[27] have reported the implementation of MP2 algorithms on distributed memory computers. Our parallel implementation is similar to the parallel direct implementation (algorithm V1) of Nielsen and Siedl[27]; however, in this study, we use a semidirect algorithm. A brief overview is presented here, but further details will be presented elsewhere.[28] For simplicity, in this study we have discussed spatial orbitals and closed-shell systems only, but these can be easily extended to open-shell systems. MP2 energy based on the RHF wave function can be expressed as:

$$E^{(2)} = \sum_{ijab} \frac{(ij|ab)^2 + \frac{1}{2}[(ia|jb) - (ib|ja)]^2}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b} \quad (3)$$

$$(ia|jb) = \sum_{\mu\nu\lambda\sigma} C_{\mu i} \cdot C_{\nu a} \cdot C_{\lambda j} \cdot C_{\sigma b} \cdot (\mu\nu|\lambda\sigma) \quad (4)$$

where C is the molecular orbitals (MO) coefficient matrix. The notations ij and ab correspond to the occupied (O) and virtual (V) molecular orbitals, which are eigenfunctions of the Fock operator with eigenvalues ε_i, ε_j and ε_a, ε_b. Ample discussion of the direct and semidirect MP2 methods have been presented in previous publications.[29–31] The next paragraph is best read with refs. 29–31 at hand.

In the Linda implementation, one of the first steps in the parallelization of the MP2 calculations involves interacting with the tuple space. Figure 2 illustrates that common blocks need to be defined in the tuple space. This is achieved by a routine called d2lin1 which functions solely to store/retrieve common blocks in/from the tuple space. In addition, d2lin2 and d2lin3 store/retrieve variables from the tuple space. The main loop (over the number of processors) checks if the master is ready to do work (single PE runs will only call the master); if not, it spawns workers to start transforming batches of integrals [see eq. (4)]. The spawning of workers is carried out by performing an *eval* operation on the linde2 routine. linde2 is the driver routine that orchestrates the computation and transformation of two-electron integrals. This basically corresponds to the algorithm shown in Figure 2 from ref. 31. In the Linda implementation, each worker gets a batch of i indices (corresponding to occupied orbitals) that need to be transformed. Each worker then generates all $(ia|jb)$ integrals for its set of i indices using one or more passes over the entire set of AO integrals.[29] Each worker requires O(OVN) of local memory to perform the integral transformation. As the $(ia|jb)$ integrals are completed, their contribution to the MP2 correction is accumulated. When all workers have completed the integrals, the partial MP2 corrections are added together via tuple space.

In the current implementation, the calculation of the two-electron integrals over AOs is carried out redundantly in both algorithms, either directly or semidirectly.

```
Store scalar variables in tuple space ( COMMON BLOCKS)
loop over PEs (P= (PEs - 1), 0, -1)
 if(P.eq.0) then
    do work on MASTER PE
  else
    eval  (LindE2)
end loop
loop over PEs - 1
  store and/or retrieve results from tuple space (arrays)
  sum up energy contributions
end loop
store or read scalar variables from tuple space
store or read array variables from tuple space
compute MP2 energy
```

FIGURE 2. Parallelization of MP2 direct and semidirect algorithms. linde2 computes $(ia|jb)$ integrals for a batch of i indices and then adds the integrals into a partial energy correction.

Performance

Performance and scalability of parallel programs depend not only on its execution time, but also on the size of the problem, parallel architecture, and number of processors. The scalability of a parallel application measures its degree of linearity with respect to the number of processors. When evaluating the parallel performance of the current implementation of Gaussian on the Cray T3E, we looked at the speedup and efficiency.

Speedup (S) is defined as the ratio of the serial run time (elapsed time, t_s) over the time that it takes to do the same problem in parallel (elapsed time, t_p):

$$S = \frac{t_s}{t_p} \tag{5}$$

Efficiency (e) is the fraction of time that a processor is doing useful work:

$$e = \frac{S}{N_{\mathrm{PEs}}} \tag{6}$$

There are many options for running Gaussian, but, in general, there are four major characteristics of a Gaussian job that determine the computational cost in a particular calculation: job type; theoretical method; basis set size; and molecular size (total number of atoms). The basic job type is to calculate either the energy or the energy and its corresponding derivatives with respect to the positions of the atoms in the molecule or with respect to things like an external electric field. This functionality can be used to optimize molecular geometries and compute the normal modes and frequencies of vibration of a molecule. Due to the importance of molecular structure in chemistry, a large majority of calculations run at supercomputer centers are geometry optimizations followed by frequency calculations. Another type of calculation that is run less often, but holds value because of its accuracy, is single-point energy calculation at high levels of theory. Any data regarding the use of these theoretical models at a typical supercom-

puter center would show the preponderance of geometry optimization in Gaussian at these sites.

We have chosen to use single-point energy (SP), FORCE (the time required for a geometry optimization is a multiple of the time needed for a FORCE calculation), and frequency calculations in this study as representative of job types being run in supercomputer centers. While it is known how to solve the time-independent Schrödinger equation in the Born–Oppenheimer approximation exactly, it is not practical to do so for all but a few small molecules due to large computational difficulties. A large number of approximate theoretical methods that range in accuracy and computational cost have been developed, and Gaussian is capable of doing most. Without discussing each theoretical method in detail (see, e.g., ref. 20), we have chosen to use Hartree–Fock (HF), the three-parameter density functional method of Becke[32] (B3-LYP), second-order Møller–Plesset theory,[20] CI singles energy and gradients[33] (MP2 and CIS), and multiconfiguration SCF (MCSCF)[33] in our study. (In order of computational cost these methods in Gaussian follow the order: HF < B3-LYP < MP2 < CIS < MCSCF.) The HF and B3-LYP methods are widely used to optimize geometries, whereas the MP2, CIS, and MCSCF methods are more often, but not exclusively, used to calculate energetics after optimization.

The following test cases were used throughout this study. Cases I–III correspond to α-pinene ($C_{10}H_{16}$). The computations for case I correspond to single-point calculations at the Hartree–Fock and B3-LYP levels of theory with 6-311G(df,p) and 6-311 + + G(3df,3p) basis sets[33] and a B3-LYP frequency calculation with 6-31G(d) basis sets (see Tables II and III). Case IV is a CI single excitations FORCE calculation using 6-31 + + G basis sets[33] on acetyl-phenol (see Table IV). A CAS(6,6) single-point energy calculation on benzene with 6-31 + G(3df) basis sets was used in case V (see Table V). MP2 single-point energy[33] was performed on a $C_{20}H_{42}$ hydrocarbon chain with 6-31G* basis sets, which corresponds to case VI (see Table VI). These molecules were chosen as small-to-intermediate size molecules to test speedup and efficiency for most of the Linda links. In case II, we also looked at effects of basis sets on scalability by enlarging the initial basis sets [6-311G(df,p)⟶6-311 + + G(3df,3p)]. All geometries (input files) may be obtained from ref. 34.

Tables II and III summarize performance for cases I–III. In the first example, a single-point energy calculation was carried out on α-pinene. It is well known that most of the time for the iterative SCF process is spent in the evaluation and use of the two-electron integrals.[10] In these particular cases, 99.5% of the time is spent in the SCF iterative scheme (`L502`). Thus, little overhead is introduced by the fact that some of the links are running sequentially. Table II shows a very good efficiency up to 32 processors. As the number of processors is increased, the amount of time spent computing the two-electron integrals decreases and the sequential diagonalization of the Fock matrix starts decreasing the parallel efficiency. The efficiency for this calculation with 64 processors is over 50%. On the other hand, by increasing the size of the basis sets for the same system (α-pinene), the efficiency is over 50% with 128 processors.

Table II also illustrates the difference in performance between the 300-MHz, 450-MHz, and 600-MHz microprocessors. Single processor improvement (from 300 to 450 MHz) is as high as a factor of 2.2. The improvement going from 450- to 600-MHz microprocessors is in the order of 20%. Clearly, improvements in the clock speed are mainly responsible for this speedup, but using higher levels of compiler optimization is also important. All runs on the Cray T3E-900 (450 MHz) and Cray T3E-1200 (600 MHz) were carried out using higher levels of compiler optimization. In addition, stream buffers (mechanism to improve off-chip memory references) were turned on as well. Anderson and coworkers[16] have already shown that higher levels of compiler optimization are important to achieve better performance.

In the case of the Cray T3E-900, it should be noted that runs using two and four processors show superlinear speedups. This is usually due to a nonoptimal single processor version of the code or to hardware features that favor multiprocessor runs. In our case, stream buffers might be responsible for the superlinear speedup seen in the Cray T3E-900 or Cray T3E-1200. This is not the case with the Cray T3E with a 300-MHz microprocessor, because the stream buffers were turned off.

Perhaps a more interesting case may be seen in Table III. The results presented in this table correspond to a frequency calculation on α-pinene. Speedups for `L502` can be obtained from Table II. Calculation of the first and second derivatives of the two-electron integrals (`L703` and `L1110`) show an almost linear speedup up to 32 processors (speedups for each link are not shown in Table III, only total speedups). This is the most expensive contribution to the derivative energy with respect

TABLE II.
Hartree – Fock and B3-LYP Single-Point Energy Calculation on α-Pinene ($C_{10}H_{16}$).

Number of PEs	L502[a]	*S*	*e*	Total[b]	*S*	*e*
HF / 6-311G(df,p)[c]						
1	18796	1.0	1.0	18861	1.0	1.0
	(8994)	(1.0)	(1.0)	(9038)	(1.0)	(1.0)
	[7405]	[1.0]	[1.0]	[7438]	[1.0]	[1.0]
2	9444	2.0	1.0	9505	2.0	1.0
	(4180)	(2.0)	(1.0)	(4225)	(2.0)	(1.0)
	[3779]	[2.0]	[1.0]	[3814]	[2.0]	[1.0]
4	4767	3.9	1.0	4829	3.9	1.0
	(2160)	(4.0)	(1.0)	(2202)	(4.0)	(1.0)
	[1953]	[3.8]	[1.0]	[1986]	[3.7]	[0.9]
8	2462	7.6	1.0	2524	7.5	0.9
	(1150)	(7.8)	(1.0)	(1192)	(7.6)	(1.0)
	[1037]	[7.1]	[0.9]	[1069]	[7.0]	[0.9]
16	1324	14.2	0.9	1386	13.6	0.9
	(639)	(14.1)	(0.9)	(679)	(13.3)	(0.8)
	[571]	[13.0]	[0.8]	[604]	[12.3]	[0.8]
32	764	24.6	0.8	825	22.9	0.7
	(379)	(23.7)	(0.7)	(420)	(21.5)	(0.7)
	[336]	[22.0]	[0.7]	[370]	[20.1]	[0.6]
64	500	37.6	0.6	562	33.6	0.6
	(249)	(36.1)	(0.6)	(291)	(31.1)	(0.5)
	[221]	[33.5]	[0.5]	[253]	[29.4]	[0.5]
B3-LYP / 6-311G(df,p)						
1	27687	1.0	1.0	27758	1.0	1.0
	(13044)	(1.0)	(1.0)	(13087)	(1.0)	(1.0)
	[10796]	[1.0]	[1.0]	[10830]	[1.0]	[1.0]
2	13869	2.0	1.0	13938	2.0	1.0
	(6097)	(2.0)	(1.0)	(6149)	(2.0)	(1.0)
	[5485]	[2.0]	[1.0]	[5520]	[2.0]	[1.0]
4	7032	3.9	1.0	7099	3.9	1.0
	(3172)	(4.0)	(1.0)	(3223)	(4.0)	(1.0)
	[2824]	[3.8]	[1.0]	[2859]	[3.8]	[1.0]
8	3665	7.6	1.0	3740	7.4	0.9
	(1692)	(7.7)	(1.0)	(1739)	(7.5)	(0.9)
	[1492]	[7.2]	[0.9]	[1524]	[7.1]	[0.9]
16	1988	13.9	0.9	2058	13.5	0.8
	(960)	(13.6)	(0.8)	(1003)	(13.0)	(0.8)
	[827]	[13.1]	[0.8]	[860]	[12.6]	[0.8]
32	1142	24.2	0.8	1216	22.8	0.7
	(579)	(22.5)	(0.7)	(622)	(21.0)	(0.7)
	[481]	[22.4]	[0.7]	[514]	[21.0]	[0.7]
64	767	36.1	0.6	835	33.2	0.5
	(400)	(32.6)	(0.5)	(443)	(29.5)	(0.5)
	[323]	[33.4]	[0.5]	[356]	[30.4]	[0.5]

(Continued)

TABLE II.
(Continued)

Number of PEs	L502[a]	*S*	*e*	Total[b]	*S*	*e*
B3-LYP / 6-311 + + G(3df,3p)[d]						
1	[96705]	[1.0]	[1.0]	[96835]	[1.0]	[1.0]
2	[49253]	[2.0]	[1.0]	[49378]	[2.0]	[1.0]
4	[25230]	[3.8]	[1.0]	[25355]	[3.8]	[1.0]
8	[13078]	[7.4]	[0.9]	[13202]	[7.3]	[0.9]
16	[6936]	[13.9]	[0.9]	[7060]	[13.7]	[0.9]
32	[3771]	[25.6]	[0.8]	[3894]	[24.9]	[0.8]
64	5546	1.0[e]	1.0	5771	1.0	1.0
	(2533)	(1.0)	(1.0)	(2685)	(1.0)	(1.0)
	[2196]	[44.0]	[0.7]	[2320]	[41.7]	[0.7]
128	3577	1.6	0.78	3804	1.5	0.76
	(1677)	(1.5)	(0.76)	(1828)	(1.5)	(0.73)
	[1420]	[68.1]	[0.5]	[1544]	[62.7]	[0.5]
256	2657	2.1	0.52	2884	2.0	0.50
	(1308)	(1.9)	(0.48)	(1459)	(1.8)	(0.46)
512	2483	2.2	0.28	2713	2.1	0.27

[a]All timings are in seconds and correspond to elapsed time.
[b]Total time to complete the run (elapsed time); numbers in parentheses are T3E-900; numbers in square brackets are T3E-1200.
[c]Total of 346 basis functions.
[d]Total of 598 basis functions.
[e]Speedup relative to 64 process elements (PE).

to nuclear coordinates. These results are consistent with previously reported high efficiencies for derivative computations.[7] The computation of derivative Fock matrices is done in the same way as the ordinary Fock matrix (L502) and shows similar scalability. However, the solution of the CPHF equations does not scale very well due to the fact that only the production of the AO integrals in the direct scheme is parallelized. The relative speedup on a Cray T3E-600 is 1.6 when doubling the number of processors from 32 to 64. For 64 processors the efficiency is over 60% for the Cray T3E-1200. Tables II and III illustrate that parallel efficiency decreases roughly in the following order: T3E-600 > T3E-900 > T3E-1200. This is not surprising, because, as the speed of the microprocessor increases (single PE performance), less parallel work needs to be done. Hence, parallel efficiency decreases. To overcome this problem the code needs to be optimized for faster processors; that is, more work needs to be distributed among processors.

TABLE III.
B3-LYP Frequency Calculation on α-Pinene.[a]

PEs[b]	L1110	L1002	L703	Total[c]	*S*	*e*
1	[17599]	[89402]	[18525]	[128814]	[1.0]	[1.0]
2	[8736]	[45224]	[9364]	[65027]	[2.0]	[1.0]
4	[4372]	[22923]	[4717]	[32856]	[3.9]	[1.0]
8	[2214]	[11880]	[2335]	[16943]	[7.6]	[1.0]
16	[1261]	[6732]	[1341]	[9645]	[13.5]	[0.8]
32	1559	9086	1369	12410	1.0	
	[668]	[3675]	[663]	[5243]	[24.6]	[0.8]
64	904	5852	765	7809	1.6	
	[369]	[2327]	[380]	[3231]	[39.9]	[0.6]

[a]The basis sets correspond to 6-31G(d).
[b]All timings are in seconds and correspond to elapsed time; numbers in square brackets are T3E-1200.
[c]Total time to complete the run (elapsed time).

Table IV summarizes elapsed timings, speedups, and total efficiency for acetyl-phenol (case IV). This case corresponds to a CI singles energy and gradients calculation. It illustrates the performance of a new link, L914, which computes excited states using CI with single excitations. It diagonalizes a matrix of the type $\langle \psi_{ia}|H|\psi_{\mathrm{HF}}\rangle$, where ψ_{HF} corresponds to the Hartree–Fock wave function and ψ_{ia} is the wave function coming from single excitations. The total CI singles energy is an eigenvalue problem. The two-electron integrals contributing to the CI singles energy can be computed by means of the PRISM algorithm. In general, we see that the speedups for L914 are consistent with L502 (not shown in Table IV). Similar to the previous case,

TABLE IV.
Elapsed Timings and Total Speedups for CI Singles[a] Energy and Gradients Calculation on Acetyl-Phenol ($C_8H_8O_2$).

PEs	L502[b]	L914	L1002	L703	Total[c]	S[d]	e
1	4641	8278	3962	1103	18014	1.0	1.0
	(2001)	(4604)	(2133)	(467)	(9235)	(1.0)	(1.0)
	[1859]	[4295]	[1641]	[435]	[8253]	[1.0]	[1.0]
4	1169	2160	1007	276	4643	3.9	1.0
	(551)	(1283)	(473)	(123)	(2461)	(3.8)	(1.0)
	[470]	[1105]	[409]	[108]	[2117]	[3.9]	[1.0]
8	614	1151	511	139	2445	7.4	0.9
	(308)	(708)	(251)	(64)	(1362)	(6.8)	(0.9)
	[274]	[646]	[229]	[59]	[1230]	[6.7]	[0.8]
16	342	631	265	71	1341	13.4	0.8
	(180)	(417)	(139)	(34)	(801)	(11.5)	(0.7)
	[160]	[370]	[126]	[31]	[709]	[11.6]	[0.7]
32	205	373	145	38	793	22.7	0.7
	(121)	(261)	(82)	(20)	(515)	(17.9)	(0.6)
	[101]	[222]	[74]	[17]	[437]	[18.9]	[0.6]
64	148	245	89	21	537	33.5	0.5
	(94)	(185)	(57)	(12)	(380)	(24.3)	(0.4)
	[78]	[154]	[52]	[11]	[318]	[26.0]	[0.4]

[a]This calculation was performed with 6-311++G basis sets.
[b]All timings are in seconds and correspond to elapsed time.
[c]Total time to complete the run (elapsed time); numbers in parentheses are T3E-900; numbers in square brackets are T3E-1200.
[d]Total speedup.

TABLE V.
Elapsed Timings and Speedups for a CAS(6,6) Single-Point Energy Calculation on Benzene (C_6H_6).[a]

PEs	L510[b]	S	e	Total[c]	S	e
1	3976	1.0	1.0	4155	1.0	1.0
	(2327)	(1.0)	(1.0)	(2505)	(1.0)	(1.0)
	[2164]	[1.0]	[1.0]	[2918]	[1.0]	[1.0]
4	1039	3.8	1.0	1219	3.4	0.9
	(652)	(3.6)	(0.9)	(825)	(3.0)	(0.8)
	[598]	[3.6]	[0.9]	[752]	[3.4]	[1.0]
8	585	6.8	0.9	763	5.5	0.7
	(371)	(6.3)	(0.8)	(546)	(4.6)	(0.6)
	[339]	[6.4]	[0.8]	[492]	[5.9]	[0.7]
16	345	11.5	0.7	521	8.0	0.5
	(228)	(10.2)	(0.6)	(403)	(6.2)	(0.4)
	[205]	[10.1]	[0.6]	[360]	[8.1]	[0.5]
32	215	18.5	0.6	395	10.5	0.3
	(144)	(16.2)	(0.5)	(144)	(17.4)	(0.5)

[a]The basis set corresponds to 6-31+G(3df).
[b]All timings are in seconds and correspond to elapsed time.
[c]Total time to complete the run (elapsed time); numbers in parentheses are T3E-900; numbers in square brackets are T3E-1200.

TABLE VI.
Elapsed Timings and Speedups for a MP2 Single-Point Energy Calculation[a] on $C_{20}H_{42}$.[b]

Number of PEs	L906[c]	*S*	*e*
1	12886	1.0	1.0
	(6573)	(1.0)	(1.0)
	[5206]	[1.0]	[1.0]
4	3799	3.4	0.8
	(2210)	(3.0)	(0.8)
	[1554]	[3.4]	[0.8]
8	2094	6.2	0.8
	(1296)	(5.1)	(0.6)
	[835]	[6.2]	[0.8]
16	1834	7.0	0.4
	(1282)	(5.1)	(0.3)
	[649]	[8.0]	[0.5]
32	1982	6.5	0.2
	(1515)	(4.3)	(0.1)
	[607]	[8.6]	[0.3]

[a]This calculation was carried out using a converged density (guess = read).
[b]Basis set corresponds to 6-31G(d).
[c]All timings are in seconds and correspond to L906 elapsed time only; numbers in parentheses are T3E-900; numbers in square brackets are T3E-1200.

first derivatives of the CI singles energy show a very good speedup up to 64 process elements (over 50% efficiency).

Finally, two single-point energy calculations were carried out at MCSCF (case V, Table V) and MP2 levels of theory. The MP2 calculations consist of a total of 384 basis functions. The range of MOs used for correlation are from 21 through 384. This gives a total of 61 occupied orbitals. In this case, increasing the number of PEs decreases the number of integrals computed per PE (or worker). In the case of 1 PE we have a total of six batches of integrals, two batches for four PEs, and finally only one batch of integrals for more than four PEs. Batches of integrals are groups of integrals based on a common molecular orbital index that can be computed at the same time.[29] Fundamentally, this parallel algorithm cannot scale past 61 (only 61 occupied orbitals in this case) processors for this case. Table VI illustrates the performance of parallel MP2 using a semidirect algorithm. The SCF calculation for this system is of the order of 1000 seconds. These data illustrate an almost linear speedup up to eight process elements. For 16 and 32 processors, the efficiency decreases to about 40–50% and 20–30%, respectively. As the number of processors increases and work gets done quicker, I/O begins to dominate the calculation. In addition, in our current implementation, the calculation of the two-electron integrals within the direct or semidirect algorithm is carried out redundantly.

The single-point energy calculation, CAS(6,6), shows a consistent speedup up to 32 process elements. On the other hand, the overall speedup is decreased due to the fact that the symbolic matrix element generator for MCSCF (`L405`) runs sequentially. Table V shows that the speedup of 12 of the 16 process elements coming from `L510` is reduced to 8, to a large extent due the overhead in `L405`.

Summary

SCF and DFT calculations with and without first and second derivatives have been shown to run efficiently on systems with up to 32 and 64 process elements. However, scaling is clearly dependent on the number of basis functions and the size of the molecule. In this study, we have examined α-pinene as an example of a molecule that shows good scaling as a function of the basis sets. Further investigation will be carried out for larger systems where cutoffs may play an important role in the number of two-electron integrals that are performed per process element.

CI singles approximation provides a direct method of computing excited states with minimal disk storage required. Due to its simplicity, CI singles provides a good approximation for large systems. In addition, in this study, we have shown that this technique is well suited for highly parallel machines. Our calculations show almost linear speedups for the links that handle the derivatives and good overall speedup throughout the entire calculation.

MP2 and MCSCF calculations, on the other hand, are more I/O demanding. In the case of MP2 calculations, the scaling is closely coupled to the number of batches distributed among all the processors. In the case presented in this study a semidirect algorithm was used. This algorithm is disk-based, and therefore most of the process elements are involved in carrying out I/O. The Cray T3E contains a large number of I/O channels, one per PE module (the T3D contains a small number of dedicated, high-bandwidth I/O nodes). We do not expect very large overhead for this particular case.

Acknowledgments

The authors thank the Corporate Computer Network at Silicon Graphics, Inc./Cray Research, for time and resources provided to carry out this study. We also thank Jim Schwarzmeier and David Slowinski for valuable discussions. We are indebted to Giovanni Scalmani for providing us with a copy of the poster that was presented at the Third National Conference on Computer Science in Chemistry.

References

1. O. A. McBryan, In *Parallel Supercomputing: Methods, Algorithms and Applications*, G. F. Carey, Ed., John Wiley & Sons, New York, p. 19.
2. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, 1991.
3. M. J. Flynn, *Proc. IEEE*, **54**, 1901 (1966).
4. M. J. Flynn, *IEEE Trans. Comput.*, **C21**, 948 (1972).
5. E. Clementi, G. Corongiu, J. Detrich, S. Chin, and L. Domingo, *Int. J. Quant. Chem.*, **18** (suppl.), 601 (1984).
6. M. Dupuis and J. Watts, *Theor. Chim. Acta*, **71**, 91 (1987).
7. R. J. Harrison and R. Shepard, *Annu. Rev. Phys. Chem.*, **45**, 623 (1994).
8. D. P. Turner, G. W. Trucks, and M. J. Frisch, In *Parallel Computing in Computational Chemistry* (ACS Series 592), T. G. Mattson, Ed., ACS, Washington, DC, 1995, p. 62.
9. J. Ochterski, C. P. Sosa, and J. Carpenter, In *1996 Cray User Group Proceedings*, B. Winget and K. Winget, Eds., Cray User Group, Inc., Shepherdstown, WV, p. 108.
10. M. D. Cooper, N. A. Burton, R. J. Hall, and I. Hillier, *J. Mol. Struct.*, **315**, 97 (1994).
11. A. Beguelin, J. J. Dongarra, G. A. Geist, R. Mancheck, and V. S. Sunderam, A *User's Guide to PVM Parallel Virtual Machine* (Technical Report ORNL/TM-11826), Oak Ridge National Laboratory, Oak Ridge, TN, 1991.
12. G. Scalmani, G. Moro, G. Cosentine, and D. Pitea, In *Third National Conference on Computer Science in Chemistry, Naples, Italy*, 1997.
13. The MPI Forum, *Proceedings of Supercomputing '93*, IEEE Computer Society, Los Alamitos, CA, 1991, p. 878.
14. R. W. Numrich, P. L. Springer, and J. C. Peterson, In *High-Performance Computing and Networking*, W. Gentzsch and U. Harms, Eds., Springer, Munich, 1994, p. 150.
15. S. Oberlin, R. Kessler, S. Scott, and G. C. Thorson, *CRAY T3E Architecture Overview*, Cray Research Manual, Chippewa Falls, MN, 1996, p. 6.
16. E. Anderson, J. Brooks, C. Grassl, and S. Scott, In *Proceedings of Supercomputing '97* (in press).
17. M. J. Frisch, G. W. Trucks, H. B. Schlegel, P. M. W. Gill, B. G. Johnson, M. A. Robb, J. R. Cheesemna, T. Keith, G. A. Petersson, J. A. Montgomery, K. Raghavachari, M. A. Al-Laham, V. G. Zakrzewski, J. V. Ortiz, J. B. Foresman, J. Cioslowki, B. B. Stefanov, A. Nanayakkara, M. Challacombe, C. Y. Peng, P. Y. Ayala, W. Chen, M. W. Wong, J. L. Andres, E. S. Replogle, R. Gomperts, R. L. Martin, D. Fox, J. S. Binkley, D. J. DeFrees, J. Baker, J. P. Stewart, M. Head-Gordon, C. Gonzalez, and J. A. Pople, *Gaussian-94*, Gaussian, Inc., Pittsburgh, PA, 1995.
18. M. J. Frisch, A. Frisch, and J. B. Foresman, *Gaussian-94 Programmer's Reference*, Gaussian, Inc., Pittsburgh, PA, 1995, p. 5.
19. J. Almlöf, K. Faegri, and K. Korsell, *J. Comput. Chem.*, **3**, 385 (1982).
20. W. J. Hehre, L. Radom, P. v. R. Schleyer, and J. A. Pople, *Ab Initio Molecular Orbital Theory*, John Wiley & Sons, New York, 1985.
21. B. G. Johnson, P. M. W. Gill, and J. A. Pople, *J. Chem. Phys.*, **98**, 5612 (1993).
22. P. M. W. Gill, M. Head-Gordon, and J. A. Pople, *J. Phys. Chem.*, **94**, 5564 (1990).
23. N. Carriero and D. Gelertner, *How to Write Parallel Programs: A First Course*, MIT Press, Cambridge, MA, 1990.
24. C. P. Sosa and N. Carriero, *CUG Spring 1997 Proceedings*, B. Winget, Ed., Fine Point Editorial Services, San Jose, CA, 1997.
25. J. D. Watts and M. Dupuis, *J. Comput. Chem.*, **9**, 158 (1988).
26. A. C. Limaye and S. R. Gadre, *J. Chem. Phys.*, **100**, 1303 (1994).
27. I. M. B. Nielsen and E. T. Siedl, *J. Comput. Chem.*, **16**, 1301 (1995).
28. C. Sosa, G. Scalmani, J. Ochterski, and M. J. Frisch (to be submitted).
29. M. Head-Gordon, J. A. Pople, and M. J. Frisch, *Chem. Phys. Lett.*, **153**, 503 (1988).
30. M. J. Frisch, M. Head-Gordon, and J. A. Pople, *Chem. Phys. Lett.*, **166**, 275 (1990).
31. M. J. Frisch, M. Head-Gordon, and J. A. Pople, *Chem. Phys. Lett.*, **166**, 281 (1990).
32. A. D. Becke, *J. Chem. Phys.*, **98**, 5648 (1993).
33. M. J. Frisch, A. Frisch, and J. B. Foresman, *Gaussian-94 Reference Manual*, Gaussian, Inc., Pittsburgh, PA, 1995.
34. All the input files used in this study can be downloaded from: http://home.cray.com/~ cpsosa/support.html or via e-mail to: cpsosa@cray.com